Individual Classifiers

Homework 2

Amber Barksdale

Machine Learning
CS-GY 6923
NYU Tandon School of Engineering
March 2022

Contents

1 Introduction to Data Set **2**

2 Naïve Bayes **2**
- 2.1 ROC & AUC . . . 2
- 2.2 Confusion Matrix . . . 3
- 2.3 Accuracy . . . 4
- 2.4 Specificity & Sensitivity . . . 4
- 2.5 Precision & Recall . . . 5

3 Support Vector Machine **5**
- 3.1 ROC & AUC . . . 6
- 3.2 Confusion Matrix . . . 7
- 3.3 Accuracy . . . 7
- 3.4 Specificity & Sensitivity . . . 7
- 3.5 Precision & Recall . . . 7

4 Decision Tree **8**
- 4.1 ROC & AUC . . . 8
- 4.2 Confusion Matrix . . . 9
- 4.3 Accuracy . . . 9
- 4.4 Specificity & Sensitivity . . . 10
- 4.5 Precision & Recall . . . 10

5 K-Nearest Neighbors **11**
- 5.1 ROC & AUC . . . 11
- 5.2 Confusion Matrix . . . 12
- 5.3 Accuracy . . . 12
- 5.4 Specificity & Sensitivity . . . 13
- 5.5 Precision & Recall . . . 14

6 Conclusion **14**
- 6.1 Models Ranked . . . 14
- 6.2 Original Paper Comparison . . . 14
- 6.3 Final Notes . . . 15

7 R Script **15**

1 Introduction to Data Set

The data set used in this analysis is called "Dry Bean Dataset", which can be found in the University of California Irvine Machine Learning Repository. The data set comes from a research paper titled Multiclass Classification of Dry Beans Using Computer Vision and Machine Learning Techniques (Koklu, M. and Ozkan, I.A., 2020)[1]. More about this dataset, the significance of dry beans in the Turkish economy, and the necessity for building an accurate bean classifier can be found in the paper itself, or the previous phase of this project titled "Exploratory Data Analysis".

The purpose of this phase of the Dry Beans data analysis project is to run a variety of multiclass classification models, compare their respective performance, and observe the tradeoffs. Four classification methods were explored: Naïve Bayes, Support Vector Machine (SVM), Decision Tree, and K-Nearest Neighbors (KNN). For each method, eight performance metrics will be discussed: Receiver Operating Characteristic (ROC) curve, Area Under Curve (AUC), confusion matrix, accuracy, specificity and sensitivity, and precision and recall.

2 Naïve Bayes

```
dt = sort(sample(nrow(drybeans), nrow(drybeans)*.7))
train<-drybeans[dt,]
test<-drybeans[-dt,]
model <- naive_bayes(train, train$Class, laplace = 1)
# PREDICT
p <- predict(model, train, type = 'prob')
```

2.1 ROC & AUC

The Receiver Operating Characteristics (ROC) curve and the area under that curve (AUC) are two metrics to check the performance of our multiclass classifier. The AUC-ROC is calculated on the test set, because that will help us estimate the general performance. The AUC tells us how capable the model is in terms of distinguishing between the classes. A higher AUC value suggests the model does a good job of predicting the class, whereas a value at 0.5 or below suggests that the classes' distributions overlap, or the model does a poor job of prediction.[2]

```
library(pROC)
multiclass.roc(test$Class,
               as.numeric(p2),
               levels=base::levels(as.factor(test$Class)),
               percent=FALSE)
```

The value of AUC for this model is **0.9665**, which suggests that this model does exceptionally well in distinguishing between bean types.

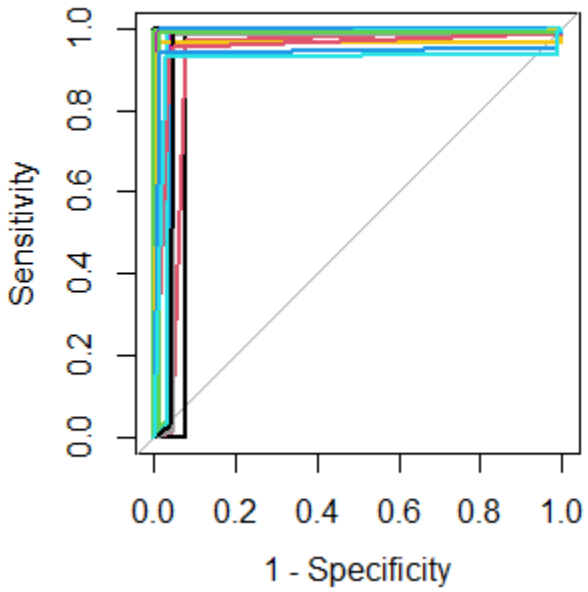

Figure 1: ROC curves for the Naive Bayes model. The Area Under the Curve (calculated for multiclass) is 0.9665.

2.2 Confusion Matrix

```
# CONFUSION MATRIX
p1 <- predict(model, train)
(tab1 <- table(p1, train$Class))

mclass1 <- 1 - sum(diag(tab1)) / sum(tab1)
mclass1
# 0.0446

TRAIN
p1          BARBUNYA BOMBAY CALI DERMASON HOROZ SEKER SIRA
  BARBUNYA       889       0   30         1     0    12    1
  BOMBAY           1     369    0         0     0     0    0
  CALI            29       0 1088         0     9     0    3
  DERMASON         0       0    0      2340    11     4   60
  HOROZ            4       0   12         2  1335     0   32
  SEKER            3       0    1        27     0  1324   11
  SIRA            24       0    3        91    17    37 1757
```

Misclassification was at approximately **4.4 %** for the training data.

```
p2 <- predict(model, test)
(tab2 <- table(p2, test$Class))

TEST
p2          BARBUNYA BOMBAY CALI DERMASON HOROZ SEKER SIRA
  BARBUNYA       343      0   15         0     1     6    5
  BOMBAY           0    153    1         0     0     0    0
  CALI            13      0  474         0     4     0    3
  DERMASON         0      0    0      1033     2     0   27
  HOROZ            3      0    3         2   542     1   12
  SEKER            2      0    0         9     0   626    5
  SIRA            11      0    3        41     7    17  720

mclass2 <- 1 - sum(diag(tab2)) / sum(tab2)
```

Misclassification was at approximately **4.7%** for the test data.

2.3 Accuracy

Accuracy was calculated as:

$$\frac{TP + TN}{TP + FP + TN + FN}$$

Where TP = True Positive, TN = True Negative, FP = False Positive, and FN = False Negative. This value was returned from the `ml_test` function.

```
BALANCED ACCURACY
BARBUNYA 0.9572453  BOMBAY   0.9998663
CALI     0.9749131  DERMASON 0.9710144
HOROZ    0.9842943  SEKER    0.9791002
SIRA     0.9541674
```

2.4 Specificity & Sensitivity

Specificity looks at all the observations that are considered "negative" - in this context, *not* a specific class - and checks how many of those were *correctly* labeled as *not* that class. In other words, it is a value that measures how well the model predicts **true negatives**. In contrast, **sensitivity** is the value that measures how well the model is able to predict **true positives**. These metrics are important because a model that is great at predicting one class may perform poorly when asked to predict a different class. Sensitivity and specificity are inversely proportional; in other words, when sensitivity increases, specificity decreases, and vice versa. A test with higher specificity means there is a lower Type-I error (false positive) rate.

```
SPECIFICITY
BARBUNYA 0.9924476       DERMASON 0.9899550
BOMBAY   0.9997325       HOROZ    0.9937685
CALI     0.9941810       SEKER    0.9951234
SIRA     0.9756923
```

Note that "sensitivity" and "recall" are calculated the same way. See the following subsection for the recall/sensitivity values.

2.5 Precision & Recall

Precision is the number of correct results or predictions divided by the number of all the returned results. A test with a higher recall/sensitivity has a lower Type-II error (false negative) rate.

```
PRECISION
BARBUNYA 0.9270270       HOROZ 0.9626998
BOMBAY   0.9935065       SEKER 0.9750779
CALI     0.9595142       SIRA  0.9011264
DERMASON 0.9726930

RECALL/SENSITIVITY
BARBUNA  0.9220430       BOMBAY   1.0000000
CALI     0.9556452       DERMASON 0.9520737
HOROZ    0.9748201       SEKER    0.9630769
SIRA     0.9326425
```

3 Support Vector Machine

A Support Vector Machine (SVM) classifier calculates the appropriate boundary, or **hyperplane** that separates the data points according to their class. The term "hyperplane" is used, because the data may be multidimensional and thus the decision boundary non-linear; the dry beans dataset certainly has more than 3 dimensions.

The optimal decision boundary is chosen for having the largest **margin** between classes - a margin being the distance between the nearest instances of the class(es) and the hyperplane. The **support vectors** are said nearest instances: the data points closest to the hyperplane.

If the distribution of the data is non-linear, then one strategy is to project the data to higher dimensions until the data are linearly separable. The SVM can then calculate a decision boundary, and eventually settle upon an optimal one. [3]

```
# SVM
library(e1071)

```

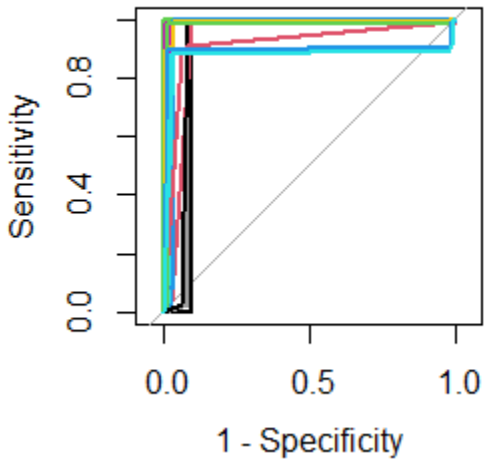

Figure 2: ROC curves for the SVM model. The Area Under the Curve (calculated for multiclass) is 0.956.

```
# remove the dependent variable from the training data set
tr <- subset(train, select=-Class)

# set the dependent variable as.factors
y <- as.factor(train$Class)

# create the SVM model
svm1 <- svm(tr, y, type="C-classification")

# Predict on the test data
te <- subset(test, select=-Class)
pred <- predict(svm1, te)
```

3.1 ROC & AUC

The ROC curves and area under the curve was calculated in the same way as for the Naive Bayes model.

```
# ROC for SVM
library(pROC)
pred_numeric <- as.numeric(pred)
mc <- multiclass.roc(test$Class, pred_numeric)

# Multi-class area under the curve:
0.956
```

3.2 Confusion Matrix

```
# CONF MATRIX
conf_matrix <- table(pred, test$Class)
conf_matrix

pred        BARBUNYA BOMBAY CALI DERMASON HOROZ SEKER SIRA
  BARBUNYA       335       0    9        0     1     4    4
  BOMBAY           0     153    0        0     0     0    0
  CALI            25       0  470        0     6     0    1
  DERMASON         0       0    0      994     2     6   67
  HOROZ            2       0   12        2   537     1   11
  SEKER            3       0    1       16     0   621   11
  SIRA             7       0    4       73    10    18  678
```

3.3 Accuracy

Accuracy was computed in the same way as the previous model - using the `mltest` library. The accuracy was reported as approximately 0.927.

```
library(mltest)
svm_metrics <- ml_test(pred, test$Class, output.as.table = FALSE)
accuracy <- svm_metrics$accuracy

ACCURACY
0.927522
```

3.4 Specificity & Sensitivity

```
library(mltest)
svm_metrics <- ml_test(pred, test$Class, output.as.table = FALSE)
recall <- svm_metrics$recall
specificity <- svm_metrics$specificity

SPECIFICITY
 BARBUNYA    BOMBAY      CALI  DERMASON     HOROZ     SEKER      SIRA
0.9948142 1.0000000 0.9904478 0.9738585 0.9914608 0.9903064 0.9652390
```

3.5 Precision & Recall

```
library(mltest)
svm_metrics <- ml_test(pred, test$Class, output.as.table = FALSE)
```

```
recall <- svm_metrics$recall
precision <- svm_metrics$precision

RECALL
 BARBUNYA    BOMBAY      CALI  DERMASON     HOROZ     SEKER      SIRA
0.9005376 1.0000000 0.9475806 0.9161290 0.9658273 0.9553846 0.8782383

PRECISION
 BARBUNYA    BOMBAY      CALI  DERMASON     HOROZ     SEKER      SIRA
0.9490085 1.0000000 0.9362550 0.9298410 0.9504425 0.9524540 0.8582278
```

4 Decision Tree

A decision tree is a very intuitive model: at any given node, based on the criteria, we move down the left or right side of the tree to the next node, until we reach a leaf that tells us what class the data point is in. The criteria at each node can be a YES/NO question, quantitative, or qualitative. The tree may not be symmetrical, and questions considering the same factor could appear at a different level than the other half of the tree. The "criteria" can also be called the "formula": the specific factors that have the most influence on the resulting class of the observation.

The decision tree formula was based on the first five principal components of this dataset. The PCA and rotation were done in the Exploratory Data Analysis project. The first five principal components are (in order): Major Axis Length, Shape Factor 2, Perimeter, Equivalent Diameter, and Convex Area.

```
library(tree)
tree.model.4 <- tree(as.factor(Class)~
MajorAxisLength+ShapeFactor2+Perimeter+
EquivDiameter+ConvexArea, drybeans)

plot(tree.model.4)
text(tree.model.4, pretty = 0, cex=0.5)

# PREDICT ON TEST SET
tree.pred <- predict(tree.model.4, test, type="class")
```

4.1 ROC & AUC

```
# ROC for Decision Tree
library(pROC)
pred_numeric <- as.numeric(tree.pred)
mc <- multiclass.roc(test$Class, pred_numeric)

```

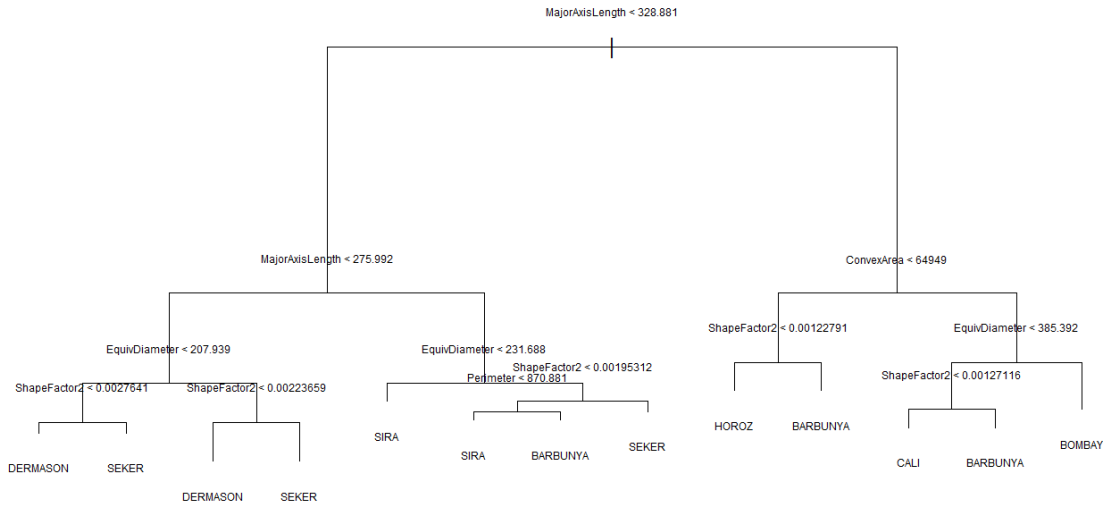

Figure 3: The decision tree formula was based on the first 5 principal components.

```
# Must plot the ROC curves individually
for(i in 1:21){
  plot.roc(mc$rocs[[i]],legacy.axes=TRUE, add=TRUE)
  sapply(1:length(mc$rocs),function(i) lines.roc(mc$rocs[[i]],col=i))
}

Multi-class area under the curve: 0.8977
```

4.2 Confusion Matrix

```
table(tree.pred, test$Class)

tree.pred  BARBUNYA BOMBAY CALI DERMASON HOROZ SEKER SIRA
  BARBUNYA      255       0   43        0    13     3   54
  BOMBAY          0     153    0        0     0     0    0
  CALI          102       0  438        0    34     0    0
  DERMASON        2       0    0      929     1    46   91
  HOROZ           2       0   13        0   468     0    5
  SEKER           5       0    0       62     0   584   15
  SIRA            6       0    2       94    40    17  607
```

4.3 Accuracy

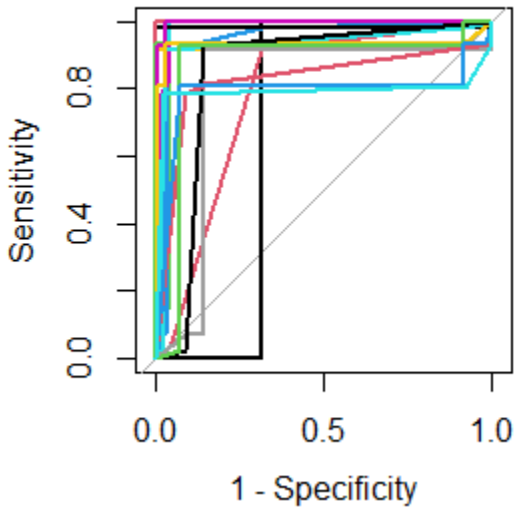

Figure 4: The ROC for the decision tree. The area under the curve calculated by the multiclass ROC was 0.8977.

```
library(mltest)
tree_metrics <- ml_test(tree.pred, test$Class)
accuracy <- tree_metrics$accuracy

ACCURACY
0.8408423
```

4.4 Specificity & Sensitivity

```
specificity <- tree_metrics$specificity

SPECIFICITY

  BARBUNYA     BOMBAY       CALI   DERMASON      HOROZ      SEKER       SIRA
0.9656744 1.0000000 0.9565773 0.9470699 0.9933021 0.9720327 0.9467515
```

4.5 Precision & Recall

```
precision <- tree_metrics$precision
recall <- tree_metrics$recall

PRECISION
```

```
 BARBUNYA    BOMBAY      CALI  DERMASON     HOROZ     SEKER      SIRA
0.6929348 1.0000000 0.7630662 0.8690365 0.9590164 0.8768769 0.7924282

RECALL

 BARBUNYA    BOMBAY      CALI  DERMASON     HOROZ     SEKER      SIRA
0.6854839 1.0000000 0.8830645 0.8562212 0.8417266 0.8984615 0.7862694
```

5 K-Nearest Neighbors

The K-Nearest Neighbors (KNN) algorithm is also relatively intuitive. Given a specific data point, the algorithm considers a handful of the nearest other data points (specifically, k other data points) and looks at the class of each of those neighbors. The majority class is chosen as the class of the current observation. The KNN algorithm accuracy depends on k, and this library automatically tests a large number of k values and returns the one with the highest accuracy.

```
9531 samples
  16 predictor
   7 classes: 'BARBUNYA',
   'BOMBAY', 'CALI', 'DERMASON',
   'HOROZ', 'SEKER', 'SIRA'

Pre-processing: centered (16), scaled (16)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 8579, 8579, 8577, 8578, 8577, 8578, ...
Resampling results across tuning parameters: (some omitted)

  k   Accuracy   Kappa
  17  0.9221820  0.9058597
  19  0.9226719  0.9064528
  21  0.9226714  0.9064450
  23  0.9230218  0.9068694
  25  0.9219026  0.9055155
  27  0.9218662  0.9054677
  29  0.9219026  0.9055133
  31  0.9216576  0.9052128
  33  0.9215886  0.9051346

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 23.
```

5.1 ROC & AUC

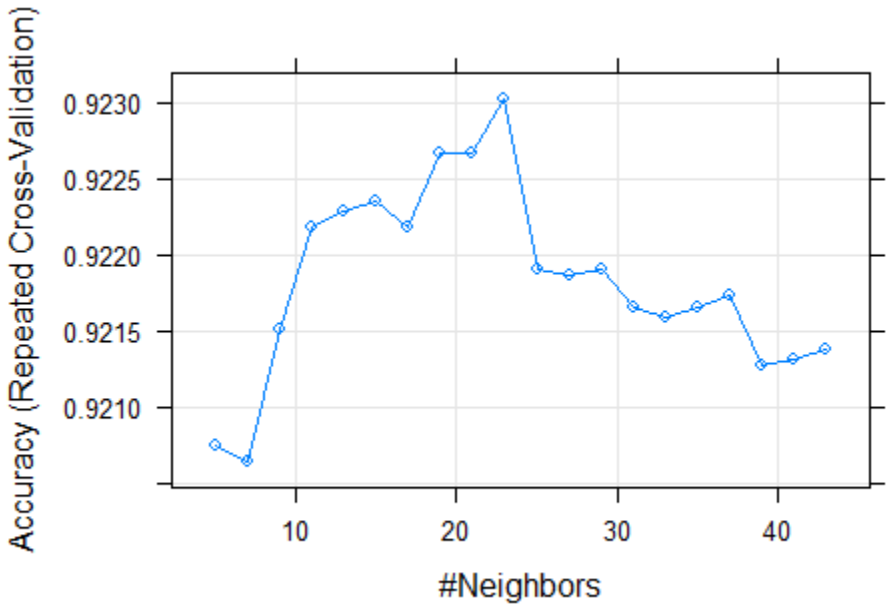

Figure 5: The accuracy of the model when using k from 5 to 45. The peak accuracy was when k = 23.

```
Multi-class area under the curve: 0.952
```

5.2 Confusion Matrix

```
Confusion Matrix and Statistics

          Reference
Prediction BARBUNYA BOMBAY CALI DERMASON HOROZ SEKER SIRA
  BARBUNYA      349       0   10        0     0     3    1
  BOMBAY          0     156    0        0     0     0    0
  CALI           30       0  464        0    14     0    1
  DERMASON        0       0    0      978     6     8   78
  HOROZ           3       0    8        2   540     0    5
  SEKER           3       0    1       24     0   575    3
  SIRA           11       0    6       59    18    22  702
```

5.3 Accuracy

```
# ACCURACY FROM CONFUSION MATRIX
Accuracy : 0.9225
```

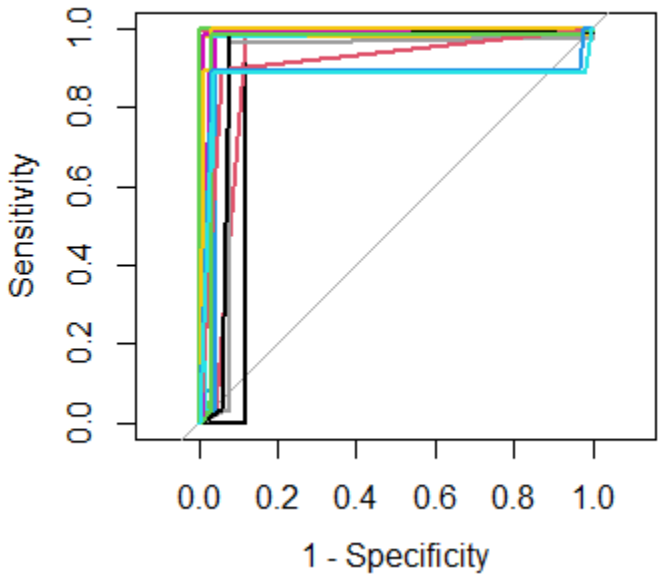

Figure 6: ROC curves for KNN when k = 23. The multiclass area under the curve = 0.952.

```
3
4 # ACCURACY FROM ML_TEST LIBRARY
5 accuracy <- knn_metrics$accuracy
6
7 0.922549
```

5.4 Specificity & Sensitivity

```
1  SENSITIVITY:
2
3  BARBUNYA       BOMBAY        CALI             DERMASON
4  0.88131        1.00000       0.9489           0.9200
5
6  HOROZ          SEKER         SIRA
7  0.9343         0.9457        0.8886
8
9  SPECIFICITY:
10 BARBUNYA       BOMBAY        CALI             DERMASON
11 0.99620        1.00000       0.9875           0.9695
12
13 HOROZ          SEKER         SIRA
14 0.9949         0.9911        0.9647
```

5.5 Precision & Recall

```
precision <- knn_metrics$precision

 BARBUNYA    BOMBAY      CALI  DERMASON     HOROZ     SEKER      SIRA
0.9614325 1.0000000 0.9115914 0.9140187 0.9677419 0.9488449 0.8581907

recall <- knn_metrics$recall

 BARBUNYA    BOMBAY      CALI  DERMASON     HOROZ     SEKER      SIRA
0.8813131 1.0000000 0.9488753 0.9200376 0.9342561 0.9457237 0.8886076
```

6 Conclusion

6.1 Models Ranked

Based on the AUC values and Accuracy values for each classifier, Naive Bayes performed the best, followed by SVM, then KNN, and finally Decision Tree.

```
AUC:                          ACCURACY:

Naive Bayes:   0.9665         Naive Bayes:   0.974
SVM:           0.955          SVM:           0.9275
Decision Tree: 0.8977         Decision Tree: 0.8408
KNN:           0.952          KNN:           0.9225
```

All the models could be further fine-tuned to improve prediction accuracy - some methods include Bagging, Boosting, and Random Forest. Of course it is always important to note that this is a relatively small data set, and the data itself may not have the same distribution as a real world data set. Intuitively, it makes sense that Decision Tree was the least accurate (although over 80% accuracy is still considered very good), because there is a lot less nuance. There may not be a straightforward binary decision for any given factor that would lead to a correct classification.

6.2 Original Paper Comparison

In the original paper, it is noted that "all models except Decision Tree have classification success with the rate of over 90%"[1]. This is in line with my results, although the original paper certainly had more rigorous testing and fine-tuning. The original paper found that their SVM model was the most accurate. They also noted that the "Bombay variety can be fully classified with 100% accuracy", which was consistent across all of my models as well [1]. In contrast, the Sira bean had the least accurate classification in both my models and the original paper. Interestingly, their Decision Tree model also used Major Axis Length as their primary/root criteria [1].

6.3 Final Notes

The full R script, including the code for Exploratory Data Analysis (the first phase of this project), can be found in the remote IBM machine or at my GitHub repository.

7 R Script

```
1
2  # HWK 2
3  library(naivebayes)
4  library(psych)
5  library(pROC)
6
7  # Starting the Naive Bayes classification
8  dt = sort(sample(nrow(drybeans), nrow(drybeans)*.7))
9  train<-drybeans[dt,]
10 test<-drybeans[-dt,]
11 model <- naive_bayes(train, train$Class, laplace = 1)
12
13 # PREDICT
14 p <- predict(model, train, type = 'prob')
15
16 # CONFUSION MATRIX
17 p1 <- predict(model, train)
18 (tab1 <- table(p1, train$Class))
19
20 mclass1 <- 1 - sum(diag(tab1)) / sum(tab1)
21 mclass1
22 # 0.0446
23
24 p2 <- predict(model, test)
25 (tab2 <- table(p2, test$Class))
26 p2
27 mclass2 <- 1 - sum(diag(tab2)) / sum(tab2)
28 mclass2
29
30 # ROC
31 mcroc <- multiclass.roc(test$Class,as.numeric(p2),
32 levels=base::levels(as.factor(test$Class)),
33                 percent=FALSE)
34
35 # Must plot the ROC curves individually
36 for(i in 1:21){
37   plot.roc(mcroc$rocs[[i]],legacy.axes=TRUE, add=TRUE)
38   sapply(1:length(mcroc$rocs),function(i)
```

```
  lines.roc(mcroc$rocs[[i]],col=i))
}

# SPECIFICITY AND SENSITIVITY
library(mltest)
classifier_metrics <- ml_test(p2, test$Class, output.as.table = TRUE)
accuracy <- classifier_metrics$accuracy
precision <- classifier_metrics$precision
recall <- classifier_metrics$recall
specificity <- classifier_metrics$specificity

# SVM
library(e1071)
train_matrix <- as.matrix(train)
# remove the dependent variable from the training data set
tr <- subset(train, select=-Class)
# set the dependent variable as.factors
y <- as.factor(train$Class)
# create the SVM model where x = the data minus DV
# and y = DV as.factors()
svm1 <- svm(tr, y, type="C-classification")

summary(svm1)

# Predict on the test data
te <- subset(test, select=-Class)
pred <- predict(svm1, te)

# Check accuracy:
table(pred, y)

# ROC for SVM
library(pROC)
pred_numeric <- as.numeric(pred)
mc <- multiclass.roc(test$Class, pred_numeric)

# Must plot the ROC curves individually
for(i in 1:21){
  plot.roc(mc$rocs[[i]],legacy.axes=TRUE, add=TRUE)
  sapply(1:length(mc$rocs),function(i) lines.roc(mc$rocs[[i]],col=i))
}

# CONF MATRIX
conf_matrix <- table(pred, test$Class)
conf_matrix

# SPECIFICITY, SENSITIVITY, PRECISION, RECALL, ACCURACY -- SVM
```

```
library(mltest)
svm_metrics <- ml_test(pred, test$Class, output.as.table = FALSE)
accuracy <- svm_metrics$accuracy
precision <- svm_metrics$precision
recall <- svm_metrics$recall
specificity <- svm_metrics$specificity

# TREE
library(tree)
tr <- subset(train, select=-Class)
# <- tree(log10(perf) ~ syct+mmin+mmax+cach+chmin+chmax, cpus)
tree.model.4 <- tree(as.factor(Class)~
MajorAxisLength+ShapeFactor2+Perimeter+EquivDiameter+ConvexArea,
drybeans)
plot(tree.model.4)
text(tree.model.4, pretty = 0, cex=0.5)
tree.pred <- predict(tree.model.4, test, type="class")

# Confusion Matrix
table(tree.pred, test$Class)

# ROC for Decision Tree
library(pROC)
pred_numeric <- as.numeric(tree.pred)
mc <- multiclass.roc(test$Class, pred_numeric)

# Must plot the ROC curves individually
for(i in 1:21){
  plot.roc(mc$rocs[[i]],legacy.axes=TRUE, add=TRUE)
  sapply(1:length(mc$rocs),function(i) lines.roc(mc$rocs[[i]],col=i))
}

# SPECIFICITY, SENSITIVITY, PRECISION, RECALL, ACCURACY -- DECISION TREE
library(mltest)
tree_metrics <- ml_test(tree.pred, test$Class, output.as.table = FALSE)
accuracy <- tree_metrics$accuracy
precision <- tree_metrics$precision
recall <- tree_metrics$recall
specificity <- tree_metrics$specificity


# K-NEAREST NEIGHBORS
library(caret)

# Split data into TEST and TRAIN
index <- createDataPartition(y = drybeans$Class,p = 0.7, list = FALSE)
train <- drybeans[index,]
```

```
test <- drybeans[-index,]

# Check distribution
prop.table(table(train$Class)) * 100
prop.table(table(test$Class)) * 100
prop.table(table(drybeans$Class)) * 100

# Preprocessing
trainX <- train[,names(train) != "Class"]
preProcValues <- preProcess(x = trainX, method = c("center", "scale"))
preProcValues

# Train
set.seed(1234)
ctrl <- trainControl(method="repeatedcv",repeats = 3)
knnFit <- train(Class~., data = train, method = "knn",
trControl = ctrl, preProcess = c("center","scale"), tuneLength = 20)

#Output of kNN fit
knnFit

# Predict
knnPredict <- predict(knnFit, newdata = test)

# CONFUSION MATRIX
confusionMatrix(knnPredict, as.factor(test$Class))

# ROC for KNN
library(pROC)
knn_pred_numeric <- as.numeric(knnPredict)
mc_roc <- multiclass.roc(test$Class, knn_pred_numeric)

# Must plot the ROC curves individually
for(i in 1:21){
  plot.roc(mc_roc$rocs[[i]],legacy.axes=TRUE, add=TRUE)
  sapply(1:length(mc_roc$rocs),function(i) lines.roc(mc_roc$rocs[[i]],col=i))
}

# SPECIFICITY, SENSITIVITY, PRECISION, RECALL, ACCURACY -- DECISION TREE
library(mltest)
knn_metrics <- ml_test(knnPredict, test$Class, output.as.table = FALSE)
accuracy <- knn_metrics$accuracy
precision <- knn_metrics$precision
recall <- knn_metrics$recall
```